Le projet Libre
LAZARUS

Outil de développement rapide d'applications

1) **LAZARUS - Développement Rapide**

2) Développement Rapide d'Application

3) Histoire : Rapid Application Development

4) Les composants et le RAD

5) LAZARUS

6) De DELPHI vers LAZARUS

7) Composants LAZARUS vs les autres

8) Création d'un composant avec ses unités

9) Création d'un composant LAZARUS

10) Comment bien créer un composant ?

11) Les Frameworks LAZARUS

1) LAZARUS - Développement Rapide d'Application

Matthieu GIROUX

- Développeur
- Exposé sur www.lazarus-components.org
- Licence Creative Common By SA

Service

- Création de Logiciels de Gestion
- Reprise de projets Open Source
- Installation et conseil

2.1) Pourquoi le Développement Rapide d'Application ?

Souvent

- On ne centralise pas assez
- Une API met du temps à être intégrée
- On doit suivre une procédure d'installation
- On fait des copiés-collés des sources
- On ne veut pas diffuser largement son code
- On manque d'indépendance

2.2) Pourquoi le Développement Rapide d'Application ?

Le DRA c'est :

- Une aide visuelle au développement
- Un code source automatisant le composant
- Surcharger des composants pour soi
- Une intégration rapide des composants
- Une installation toujours identique

3.1) Histoire : Rapid Application Development

1991 - MICROSOFT

- VISUAL BASIC puis VISUAL STUDIO

1995 - BORLAND

- DELPHI pour Windows 3.1

1999 - Projet LAZARUS

- LAZARUS - DELPHI en Open Source

2001 - BORLAND

- KYLIX - composants LINUX (abandonné)

4.1) Les composants et le RAD

Composants

- Partie réutilisable du code

Particularité

- Le composant possède des propriétés facilement manipulables

Unicité

- Gère un seul processus identifié et visible pour le programmeur

5.1) LAZARUS

C'est

- De la réutilisation et de la centralisation
- Des projets DELPHI qui deviennent libres
- Être indépendant grâce aux autres et à soi
- La possibilité de participer
- Un outil RAD nécessitant peu de mémoire

5.2) LAZARUS

Particularités

- « Write once compile anywhere »
- Multi-plateformes
- Langage PASCAL Objet
- Open Source
- Traduction à parti de DELPHI possible

5.3) LAZARUS

Avantages

- Sur WINDOWS LINUX UNIX MAC-OS BSD
- Beaucoup de composants DELPHI libres
- Exécution rapide car non retraduite (JAVA)
- Un exécutable indépendant par plateforme
- Création rapide si maitrisée

5.4) LAZARUS

Inconvénients

- Poids des exécutables important
- Jeune (Pas encore de version 1.0)
- Composants traduits ont moins de propriétés
- Utiliser les unités multi-plateformes
- Plus complet sous WINDOWS, puis LINUX

5.5) LAZARUS

Traduction DELPHI

- Simplification de l'interface DELPHI
- Code graphique à refaire
- Code système à modifier
- Des unités mixtes pour le multi-plateformes

6.1) De DELPHI vers LAZARUS

Ce qu'il faut faire

- Simplification de l'interface DELPHI
- Code graphique à refaire
- Unités LAZARUS pour le Code système
- Des unités mixtes pour le multi-plateformes
- Des unités WINDOWS si pas de bibliothèque

7.1) Composants LAZARUS vs Composants UML

Composant UML

- Module gérant un processus
- Inclus dans un package

Composant LAZARUS

- Objet visualisable avec objets et unités liés
- Inclus dans un paquet ou package qui contient les objets et unités liés

7.2) Composants LAZARUS vs Composants JAVA

Composant JAVA BEAN

- Regroupement d'objets gérant un processus
- Programmation des propriétés du JAVA BEAN

Composant LAZARUS

- Objet possédant un ancêtre et des propriétés
- Interface visuelle pour la programmation des propriétés

8.1) Création d'une unité de fonctions LAZARUS

Une unité de fonctions c'est une page contenant uniquement des fonctions

- Pour centraliser les sources
- Quand on a besoin d'une simple fonction
- Pour adapter les fonctions LAZARUS

Le nom de l'unité déterminera

- Le thème des fonctions et procédures

8.2) Création d'un composant avec ses unités LAZARUS

- Utilisation des unités pour créer un objet
- Le composant à un ancêtre visuel ou non

Avantages

- Installation rapide par le programmeur
- Création d'un savoir-faire réutilisable facilement

Inconvénient

- Création du composant

8.3) Création d'une unité de fonctions LAZARUS

Unité de fonctions : Fichier Source en programmation procédurale

Avantages

- Maintenance facile
- Centralisation du code

Inconvénient

- Nécessité de faire des copiés-collés

8.5) Création d'un composant Rappel : L'Objet

Héritage

- Le descendant hérite de son ancêtre

Méthode « dynamic »

- Surchargeable : Utiliser **inherited**

Méthode « virtual »

- Surchargeable ou remplaçable

Méthode « abstract »

- Surchargeable dans le descendant

Méthode « static »

- Non surchargeable, Un appel par classe

8.6) Création d'un composant Rappel : L'Objet

Héritage

- TFils hérite des facultés de TAncetre
- Permet de créer un composant

```
interface
type TFils = class(TAncetre)
  private
    FColor : TColor ;
  public
    constructor Create(AOwner: TComponent); override;
  published
   property Color : TColor read FColor write FColor default clBlack ;
    { public declarations }
  end;

implementation
constructor TFils.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FColor := clBlack;
end;
```

8.7) Création d'un composant Rappel : L'Objet

Polymorphisme

- Le composant possède différents formes appelées de la même manière
- Pour programmer des composants similaires

```
interface
type IDestroyInterface = interface
        procedure CallOnDestroy;
        destructor Destroy;
End;

 TExtFileCopy = class(TComponent, IDestroyInterface)
         private
           FOnDestroy : TNotifyEvent ;
         protected
           procedure CallOnDestroy ; virtual ;
         public
           destructor Destroy ; override;
         published
           property OnDestroy : TNotifyEvent read FOnChange write FOnChange;
         end;
```

8.8) Création d'un composant
Rappel : L'Objet

Encapsulation

- Le composant possède une visibilité

```
interface
 type TExtFileCopy = class(TComponent)
           private
             FOnDestroy : TNotifyEvent ;
           protected
             procedure CallOnDestroy ; virtual ;
           public
             destructor Destroy ; override;
           published
             property OnDestroy : TNotifyEvent read FOnChange write FOnChange;
           end;
implementation
destructor TExtFileCopy.Destroy ;
Begin
          CallOnDestroy;
          inherited;
End ;
procedure TextFileCopy.CallOnDestroy;
Begin
  if ( FOnDestroy <> nil ) then FOnDestroy ( Self );
End ;
```

8.9) Création d'un Composant
Les propriétés

Une propriété sert à lire et écrire sur un objet indépendamment de sa classe

Exemple: Propriétés LAZARUS

```
interface
type
  TExtFileCopy = class(TComponent)
       private
         FBeforeCopy : TEReturnEvent ;
         FBeforeCopyBuffer ,
         FOnProgress       : TECopyEvent;
         FOnChange : TEChangeDirectoryEvent ;
         FOnFailure : TECopyErrorEvent ;
         FOnSuccess : TECopyFinishEvent;
       published
        property OnBeforeCopy : TEReturnEvent read FBeforeCopy write FBeforeCopy;
        property OnBeforeCopyBuffer : TECopyEvent read FBeforeCopyBuffer write FBef
        property OnChange : TEChangeDirectoryEvent read FOnChange write FOnChan
        property OnFailure : TECopyErrorEvent read FOnFailure write FOnFailure;
        property OnProgress : TECopyEvent read FOnProgress write Fonprogress;
        property OnSuccess : TECopyFinishEvent read FOnSuccess write FOnSuccess;
       end;
```

Pas besoin d'une interface pour réutiliser les mêmes propriétés sur un autre objet !

8.10) Création d'un Composant
La déclaration published

Définition

Déclaration publique d'une classe enregistrant dans un fichier modèlisé les propriétés du composant enregistré

Le composant deviendra visualisable dans l'outil.

8.11) Création d'un Composant
La déclaration published

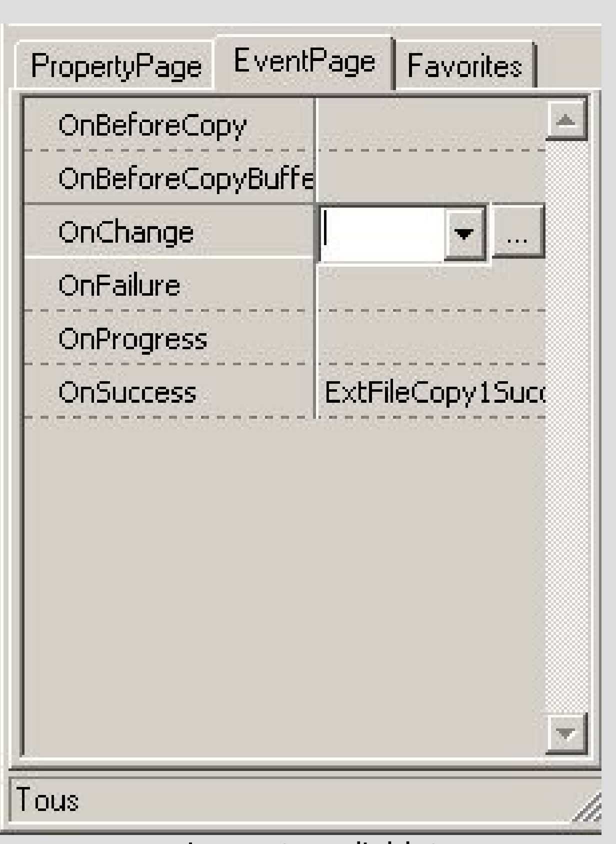
Exemple

```
interface
type
  TExtFileCopy = class(TComponent)
       private
          FBeforeCopy : TEReturnEvent ;
          FBeforeCopyBuffer ,
          FOnProgress       : TECopyEvent;
          FOnChange : TEChangeDirectoryEvent ;
          FOnFailure : TECopyErrorEvent ;
          FOnSuccess : TECopyFinishEvent;
       published
         property OnBeforeCopy : TEReturnEvent read FBeforeCopy write FBeforeCopy;
         property OnBeforeCopyBuffer : TECopyEvent read FBeforeCopyBuffer write FBefo
         property OnChange : TEChangeDirectoryEvent read FOnChange write FOnChang
         property OnFailure : TECopyErrorEvent read FOnFailure write FOnFailure;
         property OnProgress : TECopyEvent read FOnProgress write Fonprogress;
         property OnSuccess : TECopyFinishEvent read FOnSuccess write FOnSuccess;
       end;

procedure Register;

Implementation

procedure Register;
begin
 RegisterComponents('Extended', [TExtFileCopy]);
end;
```



Inspecteur d'objet

8.11) Création d'un Composant
Le mode Conception

Exemple

```
interface
type TComposant =class ( TComponent )
private
        FButton : TButton;
        procedure SetButton ( Valeur : Tbutton )
published
        property Button : Tbutton read Fbutton write SetButton ;
End;

implementation

procedure Tcomposant.SetButton ( Valeur : Tbutton );

Begin

if  assigned ( Button ) and not assigned ( Valeur ) and ( csDesigning in ComponentState ) Then
        ShowMessage ( 'Il n''y a plus de bouton....' );

if Valeur <> Fbutton Then
        FButton := Valeur;

End;
```

9.1) Comment bien créer un composant LAZARUS

Comment bien commencer ?

- L'ancêtre possède un maximum de fonctionnalités
- Faire de la recherche et développement
- Ne pas hésiter à utiliser l'objet
- Sémantique des méthodes et variables
- Créer une documentation utilisateur

9.2) Comment bien créer un composant LAZARUS

Comment faire évoluer un composant ?

- Structurer un composant
- Scinder un composant si différents objectifs
- Ne pas hésiter à utiliser les unités de fonctions

9.3) Comment bien créer un composant LAZARUS

Comment bien travailler ?

- Utilisation facile du composant
- Evolutivité
- Portabilité
- Interopérabilité avec les autres composants
- Anticipation sur la structure du composant
- Méthodes et variables en anglais adéquate

9.4) Comment bien créer une librairie LAZARUS

Chronologie de création d'un savoir-faire

- Au début on crée des unités de fonctions
- Puis on utilise et surcharge des composants
- On crée des paquets de composants
- On automatise les paquets en une librairie
- La librairie nécessite peu de code ou aucun
- On ouvre alors sa librairie aux autres API

10.1) Les Frameworks LAZARUS

Frameworks disponibles

- ZEN GL, jeux 2D sur IOS, ANDROID, WINDOWS,LINUX, MAC OSX
- Création WEB CGI par composants
- JEDI et INDY, Ensemble de Frameworks en traduction
- Framework LIBERLOG en Client/Serveur
- Logiciels Open Source avec composants

11.1) LAZARUS en 2010

Disponibilités

1)WINDOWS 32 et 64, WIN CE

2)LINUX SLACKWARE 32 et 64

3)LINUX DEBIAN 32, 64, ANDROID incomplet

4)MAC-OSX CISC et RISC (- de composants)

5)Unités UNIX pour créer à partir de LINUX

6)FPGUI LINUX, COCOA MAC avec XCODE

7)MSEGUI Crossing (peu de développeurs)

11.2) LAZARUS en 2010

Possibilités

- Programmation d'utilitaires visuels
- Accès à certains SGBD libres avec ZEOS
- Programmation WEB CGI
- Programmation de jeux multi-plateformes
- Interface Client/Serveur inaboutie
- Savoir-faire Client/Serveur LIBERLOG

11.3) Participer à LAZARUS

Si vous trouvez vos librairies dans LAZARUS

Alors pas besoin de participer

Si vous souhaitez intégrer une bibliothèque ou un composant standard, voire une plateforme en cours de développement

Alors vous pouvez les rejoindre !

LIBERLOG est à votre service...

11.4) LAZARUS en 2010

Avenir

- Ajout perpétuel de librairies de composants
- LAZARUS concurrence JAVA SE en Logiciels graphiques multi-plateformes
- LAZARUS concurrence les librairies non RAD libres en WEB avec les composants
- « On ne voit pas la fin de LAZARUS »